Interactive, Repair-Based Planning and Scheduling for Shuttle Payload Operations

G. Rabideau, S. Chien,
T. Mann, C. Eggemeyer
Jet Propulsion Laboratory
4800 Oak Grove Drive
Pasadena, CA 91109-8099
{gregg.rabideau, steve.chien,
tobias.mann, curt.eggemeyer}
@jpl.nasa.gov

J. Willis, S. Siewert
University of Colorado
Colorado Space Grant College
Campus Box 520
Boulder, CO 80309
{jwillis, siewerts}
@rodin.colorado.edu

P. Stone
Carnegie Mellon University
Computer Science Department
Pittsburgh, PA 15213-3891
pstone@cs.cmu.edu

Abstract—This paper describes the DATA-CHASER Automated Planner/Scheduler (DCAPS) system for automatically generating low-level command sequences from high-level user goals. DCAPS uses Artificial Intelligence (AI)-based search techniques and an iterative repair framework in which the system selectively resolves conflicts with the resource and temporal constraints of the DATA-CHASER shuttle payload activities.

TABLE OF CONTENTS

1. INTRODUCTION

Command sequence generation for spacecraft operations can be a laborious process requiring a great deal of specialized knowledge. Command sets can be large, with each command performing a low-level task. There may be many interactions between the commands due to the use of resources. In addition, due to power and weight limitations, the resources available on-board spacecraft tend to be scarce. Because of this complexity, tools to assist in planning and scheduling spacecraft activities are critical to reducing the cost and effort of mission operations.

This paper describes a general system that uses Artificial Intelligence Planning and Scheduling technology to automatically generate command sequences for the DATA-CHASER shuttle payload operations. The DATA-CHASER Automated Planner/Scheduler (DCAPS) architecture presented supports direct, interactive commanding, rescheduling and repair, resource allocation, and constraint maintenance.

The DCAPS search algorithm was developed based on the “iterative repair” technique used in [1]. Basically, this technique iteratively selects a schedule conflict and performs some action in an attempt to resolve the conflict. Using a repair algorithm, DCAPS is naturally well-adapted for human interaction. Therefore, the scheduler can be used as a tool to assist payload command sequencing. With the use of this tool, sequencing becomes simple enough to be accomplished by nonspacecraft and sequencing experts, such as the mission scientists. This allows the scientist to become directly involved in the command sequencing process. Following any changes in spacecraft state or user-defined goals, the repair algorithm allows simple rescheduling that avoids disrupting the original schedule as much as possible. Finally, the highly restrictive payload resources and constraints are consistently monitored and conflicts avoided automatically.

The DCAPS system is being developed for operation of the DATA-CHASER shuttle payload, which is being managed by students and faculty of the University of Colorado at Boulder. DATA-CHASER is a science payload, with a primary focus on solar observation. The main activities for the payload involve science instrument observations, data storage, communication, and control of the power subsystem. Science is performed using three solar observing instruments, Far Ultraviolet Spectrometer (FARUS), Soft X-ray and Extreme Ultraviolet Experiment (SXEE), and Lyman-alpha Solar Imaging Telescope (LASIT), that are imaging devices at various spectra.

The payload resources include power, tape storage, local memory, the three instruments, and the communication bus. DATA-CHASER is also constrained by externally-driven states such as the shuttle orientation, which affects when certain science activities can be scheduled. Payload activities must be sequenced while avoiding or resolving conflicts with resources and temporal constraints.

When using the DCAPS system, there are three modes of operation. First, by simply providing a small set of high-level science and

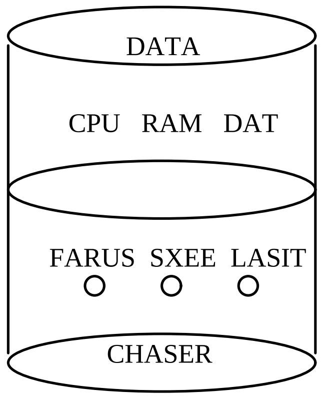

Figure 1: DATA-CHASER payload

engineering goals, an initial schedule can be generated. The goals, which describe high-level mission objectives, are automatically translated into a sequence of executable activities. The second phase offers an interactive scheduling session. Using the repair-based scheduler, the user can work with the low-level activities while maintaining consistency with resources and constraints.

After making any change in the schedule, the user can give one simple command to resolve all conflicts in the current schedule. A schedule free of conflicts, however, may not be the highest quality schedule. In the final stage, the user can call on the optimizer to generate several additional solutions based on preference information and select the best.

The main scheduling algorithm of the planner/scheduler is the repair-based search algorithm. Using this algorithm, the scheduler first collects all of the conflicts in the current schedule and classifies them based on the resource being violated and the culprit activities associated with the conflict. After choosing a conflict to repair, the scheduler must select an action to perform in an attempt to resolve the conflict. Actions include moving, adding, and deleting activities. If the action resolves the conflict, the scheduler iterates on the resulting schedule. Otherwise, the scheduler tries a different action for resolving the persistent conflict.

The remainder of this paper is organized as follows. First, we describe the DATA-CHASER shuttle payload and mission objectives. Next, we discuss the different ways in which the DCAPS system can be used to command the DATA-CHASER payload. Next, we describe the model representation. We then go into detail about the DCAPS approach to automated command generation. Then, we describe how DCAPS fits in to the overall flight and ground system architecture for the DATA-CHASER mission. Finally, we discuss related work and conclusions.

2. DATA-CHASER PAYLOAD

DATA-CHASER consists of two synergetic projects (see Figure 1), DATA and CHASER, which will fly as a Hitchhiker (HH) payload aboard STS-85 on the International Extreme Ultraviolet Hitchhiker Bridge (IEH-2) in July 1997 [2]. A technology experiment, DATA (Distribution and Automation Technology Advancement) seeks to advance semi-autonomous, supervisory operations. CHASER (Colorado Hitchhiker and Student Experiment of Solar Radiation) is a solar science experiment that serves to test DATA. The DATA technologies support cooperative operations distributed between different geographic sites as well as between humans and machines, on-board autonomy, human control, and ground automation.

CHASER is comprised of three coaligned instruments that take data in the far and extreme ultraviolet wave-lengths. The first and oldest of these instruments (17 years old) is FARUS, which takes a continuous spectrum from 115 nm to 190 nm with a resolution of .12 nm. LASIT takes images of the full solar disk of the sun in the Lyman-alpha wavelength (121.6 nm) with a Charge Injected Device imager. The final instrument in the scientific package, SXEE, consists of four photometers, each having a different metallic coating so as to enable them to look at different wavelengths between 1 and 40 nm. The objective of these instruments is to measure the full disk solar ultraviolet irradiance and obtain images of the sun in the Lyman-alpha wavelength, providing a correlation between solar activity and radiation flux as well as an association of Lyman-alpha fluxes with individual active regions of the sun.

The flight segment of the DATA-CHASER project consists of a canister that is equipped with a Hitchhiker Motorized Door Assembly (HMDA), which houses the instruments and their support electronics. The second canister contains the flight computer for the payload as well as the 2 GB Digital Audio Tape (DAT) drive that is used to store all data that is collected during the mission. The payload data is also sent to the ground system through both low rate (available 90% of the time, at 1200 bps) and medium rate (available when scheduled, at 200 kbps). The payload is also capable of receiving commands sent from the ground system when uplink is available.

During the mission, the DATA-CHASER payload will be operating in four different modes. Most of the time, when DATA-CHASER is powered, it will be in a passive mode where it is monitoring its state and notifying the ground of any changes. During the time in the mission when the orbiter is scheduled to point the bay at the sun, the DATA-CHASER payload will shift into solar active mode where all instruments take data.

The data is both written to the DAT drive on board and downlinked to the ground system for immediate data analysis. Several times during the mission, DATA-CHASER will take data while not pointing at the sun. This data is used for testing various portions of the DATA experiment with nonsolar pointing data in addition to being used for instrument calibration.

One of the consequences of flying on the shuttle system is that shuttle resources are limited, and their availability is subject to change every 12 hours. These resources include access to uplink and downlink channels, and time that your payload is allowed to operate. In addition to these resources, any given payload may also have environmental constraints as to how much contamination the payload can take. Another example is thermal constraints, such as maximum solar point time.

STS-85, the flight that DATA-CHASER payload is scheduled to fly on, is one of the most complicated flights that the shuttle has flown to date. In addition to the DATA-CHASER payload, there are four other payloads sharing the same HH bridge. In addition to the IEH-2 bridge, there is another HH bridge, a pallet payload, and a Spartan deployable satellite. Needless to say the shuttle pointing requirements are considerably tight.

In addition to modeling what the internal constraints and resources of the payload are, DCAPS must also search the shuttle flight plan for times when we are allowed to operate, downlink our data, uplink new command sets, and when we have to protect the scientific instruments from contamination events.

DATA-CHASER is an interesting scenario for scheduling because of the complex data and power management involved in the science gathering. An automated scheduler must find an optimal "data taking" schedule, while adhering to the resource constraints. In addition, the scientists would like to perform dynamic scheduling during the mission. As an example, the summary data may indicate the presence of a solar flare. If this occurs, scientists have different requirements and goals, such as higher priorities on certain instruments or longer integration times. These new goals may require a different schedule of activities.

3. User Operation

The DATA-CHASER Automated Planner / Scheduler will be part of the DATA-CHASER mission operations software. It will be a ground-based intelligent tool used for generating scheduled commands for uplink to the payload. The user's manual [3] can be found at the Jet Propulsion Laboratory. There are three phases of operating the DCAPS system: a goal satisfaction phase, an interactive repair phase, and an optimization phase.

Goal Satisfaction

The first phase of scheduling in DCAPS involves generating an initial schedule from a set of high-level, user-defined goals. The scientist or engineer simply requests one or more of the predefined goals, and the scheduler will generate the low-level activities that satisfy the goals. For example, the scientist can simply make a request for solar observations during all solar viewing periods. Given this goal, the scheduler will create and position the instrument data-take activities and their supporting activities.

Goal satisfaction is a way of generating an initial schedule. Goals are parameterized, and create activities in positions relative to certain schedule events or parameters. In this way, the same goals can be requested for different initial states. This makes them more flexible than alternate ways of creating an initial schedule, such as simply loading activities from a file. For example, the initial state in DATA-CHASER contains shuttle maneuver activities. These activities determine the solar viewing periods of the payload. The solar observation goals are based relative to these solar views, and therefore, are applicable to any maneuver sequence.

Interactive Repair

In the second phase of scheduling, the user has the opportunity for interacting with the schedule at a more detailed level. The scientist or engineer can view the activities at several levels of abstraction. The graphical user interface (GUI) can display activities from the highest level, as a single event, down to the lowest level, showing the detailed steps that make-up the activity.

The user can also modify the schedule by moving, adding, or deleting activities, as well as changing activity parameters. For example, the scientist might want to delete a LASIT data-take and replace it with a FARUS or SXEE data-take. Or, perhaps he/she may simply want to change the target of some data-take, from a solar scan to an earth scan.

Although the user has the capability of making these types of adjustments, he/she does not need to worry about the various interactions, constraints, or resource usage of the activities being modified. This information is monitored by DCAPS, and changes are propagated to the dependent objects. The results of the modification, including any conflicts, are displayed by the GUI. In addition, when the user introduces scheduling conflicts, DCAPS can resolve them automatically.

DCAPS can be called upon at any time to resolve any conflicts residing in the schedule. Conflicts are violations of resource capacities or temporal constraints. In this way, the user does not need to be very informed, careful, or specific about his/her requests. For example, a scientist can move a data-take activity without concern for its power usage. Or, a general request for data-takes can be made, without specifying the exact times for the activities to occur. Although these changes or requests may cause one or more conflicts, DCAPS can resolve these conflicts with one simple command.

Optimization

Finally, the third phase of DCAPS operation is schedule optimization. After resolving all conflicts, the schedule may still contain violations of user preferences. These preference violations can be repaired in a manner similar to repairing regular conflicts. The main difference is that the modeler must explicitly represent the types of violations and general mechanisms for repairing them. As an example, considered an engineer’s desire to have all data written to permanent storage at the end of the mission. Having data in the RAM at the end of the schedule is a violation of a preference, rather than a violation of a resource.

Preferences can also be expressed in a schedule evaluation function. In the optimization phase, DCAPS can score valid schedules based on the evaluation function developed by the modeler. One simple evaluation function may give higher scores to schedules with more science observations. Due to the fact that DCAPS utilizes a stochastic scheduling procedure, more optimal schedules can be found by simply running the scheduler many times and retaining the schedule with highest score.

4. MODEL REPRESENTATION

In order to use either Plan-IT II standalone or the full DCAPS system, the user must write a software model of the mission activities and spacecraft resources. This process involves defining a set of objects and how they interact. These definitions are then used by the scheduler to create instances of the objects.

Model Objects

The basic objects in the PI2 sequencing tool are activities, resources, slots, and dependencies.

Activities—Activities are used to model the events that happen that affect the DATA-CHASER payload, and the actions that the DATA-CHASER payload can take. All activities have some basic components: a duration, a list of slots, and a list of slot-value assignments. In addition, certain types (described below) have a list of subactivities. For these activities, the user can also define a set of temporal constraints between the subactivities. Next, we describe in more detail the four basic types of activities: events, steps, step-activities, and activities.

Events are used to model activities that do not occur in a fixed relation to other activities (e.g. Tracking and Data Relay Satellite System (TDRSS) contacts) and are not part of an activity hierarchy.

Steps are the "leaf" nodes in the activity hierarchy tree. In other words, they do not contain any subactivities. Steps cannot be instantiated without their parents and are used to model the activities at the lowest level of detail. For instance, we model an activity called CHASER-heating, which consists of two steps, CHASER-heater-on and CHASER-heater-off.

Step-activities are used to model activities at a middle level of abstraction. They can contain steps, but must also have parent activities. In DCAPS, we model an activity SXEE-Data-Take, which models the SXEE instrument opening its aperture and taking a scan. In this case, there is a step-activity called SXEE-Scan-Step, which has sensor read steps and cannot be instantiated by itself.

Activities are used to model activities at the highest level of abstraction. They are the "root" nodes in the hierarchy tree, containing subactivities, but no parent activity. The activity and event objects are what the scheduler can instantiate, and Plan-It II provides methods to access the varying levels of abstraction.

Resources—Resources define the various physical resources and the constraints they impose. Resources come in essentially five varieties: state, concurrency, depletable, nondepletable, and simple.

State resources are used to model the systems in the DATA-CHASER payload that have states associated with them. For each state resource, the modeler must specify the possible values that the state can be. Most of the systems have at least one state variable, which is whether or not they are activated. The orientation of the payload is also modeled as a state variable.

Concurrency resource constraints are used to model rules that stipulate that an activity either must occur with another activity or cannot occur with another activity. One relationship that is modeled with a concurrency resource is the requirement that a downlink or uplink can only occur during contact with a TDRSS satellite. This is modeled as a resource that is present when there is TDRSS contact activity, and required when there is a downlink or uplink activity.

Depletable resources are used to model resources with a fixed quantity, such a fuel or RAM. Activities can use some finite amount of a depletable resource, which may or may not be restorable. The amount used by the activity is persistent to the end of the schedule. In addition, the modeler must specify a maximum capacity for each depletable resource. In DCAPS, RAM is modeled as a depletable resource. Science observations produce data and use some amount of the depletable resource. Other activities, such as a transfer to permanent storage, may restore this resource.

Non-depletable resources are used to model resources which have a limit to the usage at any one time, but are reset at the end of the activity that consumes the resource. Similar to depletable resources, nondepletables are assigned a maximum capacity. Resources like power are modeled with nondepletable resources.

Simple resources are used to model devices that can only be used by one activity at a time. For instance, each of the instruments on board DATA-CHASER, FARUS, SXEE, and LASIT, are capable of taking only one image at a time and are modeled with simple resources. Simple resources are essentially nondepletable resources with an capacity of one.

Slots—Slots are parameters of activities that allow them to affect resources. They are defined separately but referenced inside activity definitions along with a value assignment for each slot. In the slot definition, the modeler must specify which resource it affects. The main types of slots are: info slots, simple slots, availability slots, choice slots, amount slots, and state slots.

Info slots are for embedding text information in activities. They are merely placeholders and do not have any effect on scheduling.

Simple slots are included in activity type definitions in order to model usage of a simple resource. For instance, there is a slot called FARUS, which is included in activity definitions of activities which use the FARUS instrument. This info slot models the usage of the FARUS instrument.

Availability slots are the slots that allow activities to provide or require the presence of a concurrency resource. There is a slot in DCAPS called TDRSS-coverage and a slot called TDRSS-coverage-needed. Both affect the TDRSS-coverage resource. TDRSS activities have the TDRSS-coverage slot, and downlink activities have the TDRSS-coverage-needed slot. TDRSS activities can be placed anywhere and provide the presence of the resource. Downlinks can only be placed in intervals where TDRSS activities are present, since this activity possesses the slot that requires the TDRSS resource to be present.

Amount slots come in two varieties: amount and reset-amount. Amount slots reduce a depletable or nondepletable resource, and reset-amount slots increase a depletable or nondepletable resource. Amount slots do not have to be associated with a resource, however. In DCAPS, we have an amount slot called Rate, which is how we model the different bit transfer rates in activities that move data, such as a downlinks or DAT reads and writes. To find the amount of data an activity transfers, we multiply the rate by the duration of the activity.

There are also two types of state slots: state-users and state-changers. State-user slots require the presence of a certain state in a state resource, and state-changer slots change the state of a state resource. The modeler must define the set of possible states. In DCAPS, there is a state resource that models the shuttle orientation, which can be solar, earth, lunar, or deep-space. Solar science activities require the shuttle orientation state to be solar, while shuttle maneuver activities change the orientation state.

Dependencies—Plan-It II provides the ability to set up links that allow one object to affect another object. These links are called dependencies. There are several types of dependencies based on the types of objects it relates: slot-to-resource, slot-to-slot, slot-to-activity start or duration, activity start or duration-to-slot, and resource-to-resource dependencies.

Slot-to-resource dependencies are the default dependencies in the Plan-It II system. They allow a slot to affect a resource, and are created automatically when a slot is defined with the same name as a resource.

Slot-to-slot dependencies allow the value of one slot to affect the value of another slot. For instance, in the DAT-transfer activity, there are two slots, one that models the removal of data from the RAM, and one that models the addition of data to the DAT (digital tape). In DCAPS, a dependency has been defined that sets the value of one of the slots equal to the value of the other slot (so that the amount subtracted from RAM is never different than the data added to the DAT).

Activity start time or duration-to-slot dependencies and slot-to-activity start time or duration dependencies facilitate the modeling of convenient relationships among Plan-It II objects. For instance, the DAT-transfer has a slot called Rate, which is the rate at which data can be moved from the RAM to the DAT. We have a dependency that sets the amount of data that is removed from the RAM equal to the rate multiplied by the duration. An example of a dependency that goes from slot to duration is a dependency which links the selected target for a science image to the length of time it takes for the instrument to scan. The duration of a FARUS scan varies depending on its use of the shuttle orientation state (solar, earth, or lunar).

Resource-to-resource dependencies allow one resource to affect another resource directly. This is very convenient for modeling power usage, since power consumption can be tied to activities or states. For instance, power consumption by the heater can be linked to an activity (e.g., the activation of the heater), or to a state of the heater (e.g., when the state of the heater is “on,” more power is used).

Hierarchy

The modeler can create an activity hierarchy when defining the activities. All this means is that activities can have subactivities which can also have subactivities, and so on. Only the activity at the top of the hierarchy can be instantiated in the schedule. When an activity is created, all of its subactivities are created automatically. Therefore, the scheduler must schedule the entire hierarchy if it wants to schedule one of the components.

In modeling the DATA-CHASER shuttle payload, decisions had to be made about where to put activities in the activity hierarchy. We decided to model those activities that could be scheduled arbitrarily (and had no subactivities) as events not in a hierarchy. Some activities that were modeled as events were TDRSS contacts, shuttle venting, and very simple activities that could occur independently, like relay activations and HMDA operations (opening and closing).

If one event always occurred in some fixed temporal relationship to another, then we modeled it as an activity in a hierarchy. For instance, a SXEE data-take consists of a number of calibrations, a door opening activity, several scans, a door closing activity, then a data transfer to the RAM buffer. We modeled all of these activities as steps in an activity called SXEE-Data-Take.

Common Strategies

There were a number of strategies that we employed in the modeling process that made modeling the DATA-CHASER payload simpler.

One strategy that we employed was to reduce the number of states that state variables could have through discretization. For instance, spacecraft orientation is best modeled with a real valued 3 dimensional vector. But for modeling purposes, we reduced the number of possible orientations to a discrete set of four: solar, lunar, earth, and deep space.

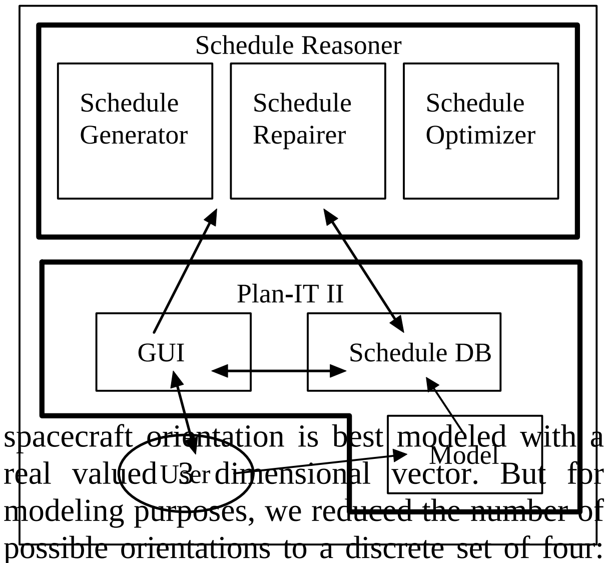

Figure 2: DCAPS architecture

Another strategy that we employed in modeling DATA-CHASER was to separate one component into several. For instance, there was really only one memory buffer that was used for storing several types of data, but we modeled it as though it were three buffers: one for science data, one for engineering data, and one for storing data to be downlinked. We also did this with power. There are really only two power sources, DATA power and CHASER power, but we modeled them as though there were different power resources for each of the science instruments and several of the other subsystems. This allowed us to track power usage more conveniently.

5. Automated Planner/Scheduler

The DATA-CHASER Automated Planner / Scheduler produces a complete, valid schedule of payload operation commands from a model, initial state, and set of high-level goals. In addition, it can input intermediate, invalid schedules (resulting from user changes) and produce a similar, but valid schedule. Finally, the scheduler can take several valid schedules, score them, and select the most optimal schedule.

The planner/scheduler consists of two main parts, the Plan-IT II (PI2) sequencing tool [4] and the schedule reasoner (see Figure 2). PI2 was written by William C. Eggemeyer and originally designed as an “expert assistant sequencing tool.” PI2 includes a GUI that allows for easy manipulation of the schedule. In addition, it serves as an activity/resource database that supplies valuable information to the schedule reasoner. PI2 supports complex monitoring and reasoning about activities and the various constraints between them. The schedule reasoner uses Artificial Intelligence (AI) techniques to automatically generate new schedules, repair existing faulty schedules, and optimize valid schedules. PI2 provides information about resource availability and conflicts; the scheduler must decide which activities to use to resolve the conflicts and where to place the activities temporally. The two components work together to provide easy and fast sequencing of mission activities.

Schedule Data-Base

In the DCAPS system, PI2 is used primarily as a “schedule database” and resource constraint checker. It was originally developed as a graphical sequencing tool. Activities and resources are displayed on a graphical output. An activity represents some mission event that occurs over a period of time and uses some of the mission resources. A resources represents some limited available material whose usage is modeled as discrete blocks over time.

For each type of activity and resource, PI2 displays a timeline, which represents the behavior of that activity/resource type over a period of time. When activities are created, they are placed at a specified time on the timeline. Resources used by that activity are updated to reflect the additional usage. In addition to schedule visualization, PI2 provides an easy-to-use input interface for modifying the schedule. Moving activities is as simple as a click-and-drag with a mouse.

PI2 helps ease the burden on sequencers by continually monitoring all activities in the sequence. As activities are added or moved, the change in resource usage is automatically updated, and the new resource profiles are displayed. With this information available, the user can immediately see the effects of a schedule change on the mission resources. For each resource, PI2 also monitors any conflicts that are occurring on the resource.

Conflicts are time intervals where the limitations of the resource have been exceeded. These conflict intervals are highlighted in red to flag their existence for easy identification. Finally, PI2 monitors any dependencies that have been defined between activities and resources. The values of specific parameters of activities and resources may be functionally dependent on values of other parameters. PI2 automatically keeps these parameter values consistent.

PI2 also helps out by serving as an activity and resource database, producing/accepting information to/from a sequencer. The functional interface to PI2 has been extended to better assist an automated sequencer. A basic set of “fetch” functions have been developed to quickly retrieve information about conflicts and the resources and activities involved in the conflict. For example, an interface function has been written to fetch the legal times where an activity can occur in the schedule. Here, “legal times” refers to positions where no conflicts are caused by any of the resources used by the given activity.

In addition to fetching information about the current state of the schedule, the user will need to be able to change the current state in attempt to fix or optimize the schedule. Some basic primitive functions are provided by PI2 to allow an external system to add and move activities, change their duration, etc. These primitives make up the set of actions that a scheduler can take when trying to resolve conflicts.

Schedule Reasoner

The second major component of DCAPS is the automated schedule reasoner. This is the next step in automating and simplifying the spacecraft command sequencing process. There are three parts to the schedule reasoner: a schedule generator, a schedule repairer, and a schedule optimizer. First, the schedule generator will transform a set of user-defined, high-level goals into a valid sequence of low-level commands. Second, the schedule repairer will automatically restore the consistency of the sequence after arbitrary user interaction by rescheduling using repair actions. The scheduler repairer iteratively attempts to resolve each conflict, which involves making choices on what to repair and how to repair it. Finally, the schedule optimizer can optimize a valid schedule to increase the scientific return.

Schedule Generator—The first step in sequencing spacecraft commands is to come up with an initial schedule of events for each phase of the mission. This process has been partially automated in DCAPS with the schedule generator. Expressing schedules and partial schedules to be generated is done through user defined goals. There are two ways in which user goals are handled in DCAPS. First, initial science and engineering goals are handled with parameterized scheduling functions. Each function implements a goal. For example, there is a "Place-Power" function that schedules power switching activities in appropriate places based on some engineering parameters. Parameters may include such things as a minimum time between switching, or a power on during a particular state of a different resource.

Second, science goals can also be expressed through data-take requests, which do not have to be a part of the initial schedule generation. For example, a scientist can request ten additional scans from a particular instrument to occur any time during some phase of the mission. This type of general request does not include specific locations or necessary supporting activities. The scheduler will simply place them at random positions and allow any conflicts to be resolved by the automated repairer.

Schedule Repairer—The generated initial schedule may still violate some of the spacecraft constraints. Also, the scientists and engineers might feel that their goals were not completely satisfied, and may need to interact with and modify the generated schedule. By modifying the schedule, new conflicts may be introduced. Therefore, we need some way of automatically resolving any existing conflicts in the schedule, while disrupting the current state of the schedule as little as possible. Having the process automated allows the user to be less careful, and therefore spend less time on the details of sequencing the activities. When general requests or changes have been made, all conflicts can be resolved by executing one simple command to invoke the schedule repairer.

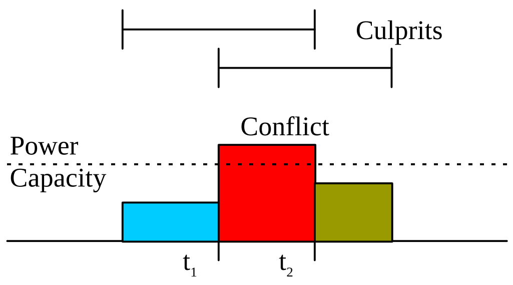

Figure 3: Conflicts

Before describing the schedule repairer, we must present a few definitions. A "hard conflict," or just "conflict," is a violation of one of the resource constraints. A conflict occurs over a certain time period and is caused by activities called "culprits." For example, if the power capacity is exceeded from time t_1 to time t_2, then a conflict exists from time t_1 to time t_2, and the culprits are any activities that use power during this time (see Figure 3). A "soft conflict" is a violation of one of the user's high level goals. "Hard conflicts" are violations of legal constraints, while "soft conflicts" are violations of user preferences. "Choice points" are places in the scheduling algorithm when a decision must be made. For example, when there are many conflicts to resolve, the scheduler must decide which conflict to resolve first. A "hard choice," or just "choice," is a decision made solely on the basis of possible hard conflicts. It may be decided, for example, not to place an activity at a certain time because new conflicts will be added as a result of that placement. A "soft choice" is a decision made on the basis of user preferences or heuristics with the hopes of generating a more optimal schedule. An example of a user preference is a priority scheme on certain activities. One heuristic may be to move lowest-priority culprits to the nearest legal position.

There are three possible actions to take in attempt to resolve a conflict: move, add, or delete an activity. The “move” action involves moving one of the culprits of the conflict to a positions that will either resolve the conflict or at least ensure that the moved activity is no longer a culprit. Some conflicts can be resolved by adding a new activity. These activities usually provide some resource that was previously not available. Finally, a conflict can also be resolved by simply deleting the culprits. This is obviously not a preferred method and is only used as a last resort.

The resolution of a conflict greatly depends on the type of resource that is in violation. There are five different types of conflicts corresponding to the five types of resources. A state conflict occurs when an activity requires the resource to be in a state which it is not. The culprits in this type of conflict are all of the activities that require the incorrect state and the activity that changed the resource to the incorrect state. Several possibilities for resolving a state conflict include moving the culprits to another interval where the required state is present or adding an activity that will change the state of the resource to the required state.

A concurrency conflict is when an activity requires the presence of the resource during a time for which it is absent. The culprits in this type of conflict are all of the activities that require the presence of the resource. To resolve a concurrency conflict, the scheduler can move the culprits to an interval where the resource is present or add an activity that provides the presence of the resource.

A depletable conflict means that the activities of the schedule have used too much of the resource. In this type of conflict, the culprit is the activity that caused the resource to overflow during the time that it first overflows. Some depletable resources have “resetter” activities and this sort of conflict can be resolved by adding an activity that “resets” the available resource. For example, a downlink activity will free up space in the downlink buffer. A nondepletable conflict is when activities overuse a resource during a particular time interval. The culprits in this type of conflict are all of the activities that use the resource during the conflict interval. This sort of conflict can be resolved by moving or deleting culprits. There are no activities in the DATA-CHASER model that can add to a nondepletable resource.

Simple conflicts occur when two or more activities use the same resource at the same time. This type of conflict can only be resolved by moving culprits.

For any type of initial schedule, the schedule repairer must find the correct activities to move, add, or delete and position them temporally in such a way that no conflicts remain. The scheduler makes decisions randomly except at certain choice points where heuristics are used. The scheduler relies on some interface functions to PI2 that describe the conflicts in the current schedule, describe the activities that could resolve a conflict, and manipulate the schedule. We first describe the random scheduler, followed by the heuristic enhancements that facilitate scheduling within the DATA-CHASER domain. The ultimate task

Iterative Repair Algorithm

The following is the algorithm for the schedule repairer written in a C-like pseudo-code.

```
Resolve-Conflicts ()
{
   iterations = 1
```

```
    conflicts = GetConflicts()
    Loop while (length(conflicts) > 0 && iterations <= max-iterations) {
       conflict = ChooseConflict(conflicts)
       method = ChooseMethod(conflict)
       case (method) {
       ‘move’
          culprit = ChooseCulpritToMove(conflict)
          duration = ChooseDuration(conflict, culprit)
          start-time = ChooseStartTime(conflict,culprit,duration)
          success = MoveCulprit(conflict,culprit,start-time)
       ‘add’
          activity = ChooseActivityToAdd(conflict)
          duration = ChooseDuration(conflict, activity)
          start-time = ChooseStartTime(conflict,activity,duration)
          success = AddActivity(conflict,activity,start-time)
       ‘delete’
          culprit = ChooseCulpritToDelete(conflict)
          success = DeleteCulprit(conflict,culprit)
       }
       progress = GetProgress()
       if not(success || progress) then UndoLastAction()
       conflicts = GetConflicts()
       iterations = iterations + 1
    }
}
```

of the system is to find the best place to schedule the activities so as to maximize the utility of the schedule. In the basic scheduler, all choices are made randomly from the list of options unless otherwise specified.

The algorithm is a simple iterative loop over the conflicts in the schedule. First, a conflict is selected from the list of current conflicts. An attempt is made to resolve the chosen conflict. Next, a method for resolving the conflict is chosen. The repair action will depend on which method has been selected. If “move” is chosen, then a culprit must be picked from the list of culprits in the conflict. A duration and start time are chosen for the culprit, and the culprit is moved to the new location. If “add” is the chosen method, then the repairer must decide which activity type to instantiate. Again, a duration and start time must be chosen for the new activity, and the activity is inserted at the chosen time. If the repairer chooses to “delete” an activity, then it simply must choose an activity to delete, and delete it. After the chosen action is performed, the schedule repairer checks to see if progress was made. We define progress as either decreasing the number of conflicts, decreasing the number of culprits, or decreasing the duration of the conflicts.

If the action did not succeed in resolving the conflict, or progress was not made, then the action is “undone.” Otherwise, the new set of conflicts are found, and the loop counter is incremented. This process continues until all conflicts are resolved, or the loop counter exceeds a user-defined maximum bound. For every choice point in the algorithm, where a selection must be made from a list of possibilities, the schedule repairer is allowed to backtrack to that point. What this means is, that if a particular choice fails, the schedule repairer may choose another from the list before giving up. If all choices fail, then a previous decision must have been incorrect, and the repairer can backtrack to the preceding choice point. All choice points, including the decision on whether or not to backtrack, are heuristic decisions and may customized to a particular domain.

Schedule Optimizer—The schedule optimizer is composed of additional knowledge supplied by the user and utilized by the other components of the scheduler. There are three ways to optimize a schedule: using preference heuristics at search choice points in the schedule repairer, specifying a set of "soft conflicts" for the repairer, and using an evaluation function to score results from multiple runs of the schedule generator and repairer.

A preference heuristic, or "soft choice," can be made at any decision in the repair search. For example, when deciding where to move a conflict causing activity, the user might prefer to move that activity to a position closest to its current position. This will help the scheduler avoid unnecessary disruption to the existing schedule. The existing schedule, after all, may have been produced by the user in an attempt to optimize the schedule.

Preferences can also be expressed using what we referred to as "soft conflicts." A soft conflict is a way of specifying a preferred value for a particular resource, possibly at a particular time. For example, having any scanned data that has not been stored on the tape at the end of the mission, is considered a soft conflict. This is not a hard conflict, because the data is not exceeding the buffer size. However, the scientist would prefer that all of the data be written to the tape at the mission's end, rather than leaving it in the on-board memory. After the schedule repairer handles all of the hard conflicts, it continues by iteratively addressing all of the soft conflicts.

The third approach to optimization involves scoring several resulting schedules and choosing the one with the highest score. The evaluation function is domain dependent and would have to be written separately for each application. Some basic scoring, however, will be similar across applications. For example, most science spacecraft are mainly concerned with collecting the largest number of images as possible. A simple evaluation would give a higher score to schedules with greater amounts of collected data. Once we have the evaluation function, we need to be able to produce several different schedules from the same goals and initial state.

This can be done by either changing the heuristics or by running the scheduler with a different random seed. Some heuristics may work better than others, and it is often difficult to tell which is the best for a particular application. Therefore, it may be necessary to resort to empirical tests. After running the scheduler on different heuristics, we can simply choose the set of heuristics which generates the schedule with the highest score. After choosing the heuristics, the scheduler can be run many times with different random seeds. At choice points where there is no heuristic for choosing from the list of possibilities, the scheduler makes a random decision. With different random seeds, these decisions will be different, and the resulting schedule will be different. Using the evaluation function, we can assign a score to each, and choose the schedule with the highest score. This procedure will not necessarily uncover the optimum schedule, but it will help find a more optimal schedule.

Heuristics—The general search and decision making described above would be futile without expert support and guidance. Heuristics have been developed and incorporated into DCAPS to help guide the search to a valid and more optimal schedule. This guidance knowledge comes from both domain experts and scheduling experts. There are three basic classes of heuristics used in DCAPS: selection, pruning, and backtracking heuristics.

Selection heuristics involve deterministically sorting or selecting from a list of possibilities at a choice point in the search. The selection is usually based on some property of the objects being considered. For example, when choosing a culprit to move in order to resolve a power conflict, one heuristic might choose the culprit that uses the most amount of power. Using this heuristic might resolve the conflict faster. Another successful heuristic used in DCAPS was one that sorted the possible locations for activity placement by the number of conflicts the activity would cause when placed in that location. This basic approach has been referred to as the "min-conflicts" heuristic [5]. The min-conflicts algorithm we use is interesting, and it is worthwhile to go into detail.

For each resource used by an activity, we query the database for the legal times where the activity can be placed without violating the resource constraint. Then, each legal interval is assigned an initial score of one. Next, we intersect two sets of intervals that resulted from two of the resources, using a special "scored" interval intersection (see Figure 4). The scored intersection of intervals A and B results in four possibilities: an interval with a score of A for positions where A exists and B does not, an interval with a score of B where B exists and A does not, an interval with a score of A plus the score of B where the two intervals intersect, or no interval where neither A nor B exist. The result of this intersection is then intersected with the third set of intervals.

This process continues until each set of intervals for each resource has been intersected. The result is a set of scored intervals, where the score represents the number of resources that will not be violated if the activity is placed in that position. Using these intervals, we can choose a position with the highest score, in other words, the position with the fewest conflicts.

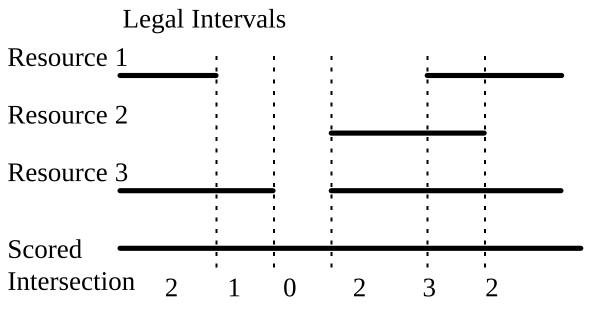

Figure 4: Min-conflicts with scored interval intersection

Another class of heuristics used in DCAPS are the pruning heuristics. These heuristics remove some of the possibilities for a given selection in attempt to make the choice easier and faster. For example, after finding the scored intervals for an activity, we may not want to try all possible positions. One possibility is to only try positions with the highest score or least number of conflicts. This process may speed up scheduling because the scheduler will only try a few positions before realizing this attempt is futile and giving up to try something different. Too much pruning, however, may remove possibilities that could be useful. In the above example, some of the pruned intervals may have included positions that, if the activity was placed there, would have improved the schedule. A more conservative approach might be to prune only those intervals that would cause more conflicts than are currently in the schedule. These intervals cannot possibly be positions that could improve the schedule.

Finally, backtracking heuristics are used to help determine when to continue working on the same problem and when to move on to a different problem. At each choice point, we have a list of possibilities. If we try one possibility, and it fails, we can continue and try the next possibility, or move on to a different choice point. Heuristics can be used to help make two types of decisions about backtracking: deciding on “action failures” and deciding on “selection failures.” First, the notion of an “action failure” is not clear and requires an approximate definition. Success is not simply resolving the chosen conflict. When, resolving a conflict, and action attempt may fix the chosen conflict, but cause several other conflicts.

Therefore, success can be thought of as improving the schedule. But how much? And what defines an improvement? Our current definition of progress includes observing the change in the number of conflicts, the change in the number of culprits, and the change in the duration of the conflicts. Checking the progress of an action can be used as a heuristic for determining whether to accept the action, or try a different one. The second opportunity for heuristics comes when deciding if there is a “selection failure.” While trying and failing on a list of possibilities for a choice point, at some point we must decide that the previous choice was a failure. Heuristics can help with this decision also.

6. System Integration

DCAPS will be integrated into the End-to-End Mission Operations System (EEMOS) that is currently being developed for the DATA-CHASER project as a prototype for the Pluto Express EEMOS [6]. Currently the DATA-CHASER EEMOS consists of seven parts: Command and Control, Fault/Event Detection Interaction Reaction (F/EDIR), DATA/IO (Data handling), the Ground Database, the Graphical User Interface, the software testbed, and finally the planning and scheduling system (DCAPS).

The command and control system that we are using, System Command Language (SCL, also known as Spacecraft Command Language), integrates procedural programming with a real-time, forward-chaining, rule-based system. DCAPS interfaces with SCL through DATA/IO by sending script scheduling commands that are to be scheduled either on the flight or ground system. This interface is implemented by mapping PI2 activities to SCL scripts that were written prior to flight and can be scheduled or event-triggered by activating rules. These scheduling and rule activation commands are then sent to DATA/IO which forwards that list to the SCL Compiler. Once compiled, the list is sent to the payload through the next available uplink.

DCAPS is also interfaced with the ground EEMOS database, O2. O2 is an object-oriented database that will be used to store all mission data and telemetry that is downlinked by the payload. It will also store a command history. Through DATA/IO, DCAPS will request current payload status data in the form of sensor values in the telemetry history. It will also request lists of all commands uplinked during a given time interval. These are used by DCAPS to infer command completion status as well as to get the current state of the payload so that a new schedule can be created.

During mission operations, approximately every four hours or so, DCAPS will be asked by an operator to generate script scheduling commands and rule activations for the next six hours according to its schedule. Once this list is finished, it is reviewed by the Mission Operations staff on duty. If judged to be correct, scheduling and rule activation commands will be sent to DATA/IO during the next available uplink window.

If during that six hour period there is a major change in the NASA activities, DCAPS will ask if the users want to update the schedule script on-board. Due to the fact that SCL currently has no scheduled script instance identification, this will involve descheduling all remaining scripts in the queue and then rescheduling them. This is acceptable if the user did not schedule any scripts independently of DCAPS. If he/she did, and DCAPS reschedules its list, the user's scheduled commands will be lost. If the user accepts it, DCAPS will generate a updated list, ask the user to verify it, and then deschedule rest of the old list and schedule the new list. Future versions of SCL will most likely support scheduling instances, therefore alleviating these problems.

7. Summary and Related Work

Iterative algorithms have been applied to a wide range of computer science problems such as traveling salesman [7] as well as Artificial Intelligence Planning [8,9,10,11]. Iterative repair algorithms have also been used for a number of scheduling systems. The GERRY/GPSS system [1,12] uses iterative repair with a global evaluation function and simulated annealing to schedule space shuttle ground processing activities. The Operations Mission Planner (OMP) [13] system used iterative repair in combination with a historical model of the scheduler actions (called chronologies) to avoid cycling and getting caught in local minima. Work by Johnston and Minton [5] shows how the min-conflicts heuristic can be used not only for scheduling but for a wide range of constraint satisfaction problems. The OPIS system [14] can also be performing iterative repair. However, OPIS is more informed in the application of its repair methods in that it applies a set of analysis measures to classify the bottleneck before selecting a repair method.

In summary, DCAPS represents a significant advance from several perspectives. First, from a mission operations perspective, DCAPS is important in that it significantly reduces the amount of effort and knowledge required to generate command sequences to achieve mission operations goals. Second, from the standpoint of Artificial Intelligence applications, DCAPS represents a significant application of planning and scheduling technology to the complex, real-world problem of spacecraft commanding. Third, from the standpoint of Artificial Intelligence Research, DCAPS mixed initiative approach to initial schedule generation, iterative repair, and schedule optimization represents a novel approach to solving complex planning and scheduling problems.

Acknowledgments

This work was performed by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

References

[1] M. Zweben, B. Daun, E. Davis, and M. Deale, "Scheduling and Rescheduling with Iterative Repair," in *Intelligent Scheduling*, Morgan Kaufman, San Francisco, 1994.

[2] DATA-CHASER Documents, Annual Report.

[3] G. Rabideau, S. Chien, T. Mann, C. Eggemeyer, P. Stone, and J. Willis, "DCAPS User's Manual," JPL Technical Document D-13741, 1996.

[4] W. Eggemeyer, "Plan-IT-II Bible", JPL Technical Document, 1995.

[5] M. Johnston and S. Minton, "Analyzing a Heuristic Strategy for Constraint Satisfaction and Scheduling," in *Intelligent Scheduling*, Morgan Kaufman, San Francisco, 1994.

[6] S. Siewert and E. Hansen, "A Distributed Operations Automation Testbed to Evaluate System Support for Autonomy and Operator Interaction Protocols," *4th International Symposium on Space Mission Operations and Ground Data Systems*, ESA, Forum der Technik, Munich, Germany, September, 1996.

[7] S. Lin and B. Kernighan, “An Effective Heuristic for the Traveling Salesman Problem,” *Operations Research Vol. 21*, 1973.

[8] S. Chien and G. DeJong, "Constructing Simplified Plans via Truth Criteria Approximation," *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems*, Chicago, IL, June 1994, pp. 19-24.

[9] K. Hammond, “Case-based Planning: Viewing Planning as a Memory Task,” Academic Press, San Diego, 1989.

[10] R. Simmons, “Combining Associational and Causal Reasoning to Solve Interpretation and Planning Problems,” Technical Report, MIT Artificial Intelligence Laboratory, 1988.

[11] G. Sussman, “A Computational Model of Skill Acquisition,” Technical Report, MIT Artificial Intelligence Laboratory, 1973.

[12] M. Deale, M. Yvanovich, D. Schnitzius, D. Kautz, M. Carpenter, M. Zweben, G. Davis, and B. Daun, “The Space Shuttle Ground Processing System,” in *Intelligent Scheduling*, Morgan Kaufman, San Francisco, 1994.

[13] E. Biefeld and L. Cooper, “Bottleneck Identification Using Process Chronologies,” *Proceedings of the 1991 International Joint Conference on Artificial Intelligence*, Sydney, Australia, 1991.

[14] S. Smith, “OPIS: A Methodology and Architecture for Reactive Scheduling,” in *Intelligent Scheduling,* Morgan Kaufman, San Francisco, 1994.

***Gregg Rabideau** is a Member of the Technical Staff in the Artificial Intelligence Group at the Jet Propulsion Laboratory, California Institute of Technology. His main focus is in research and development of planning and scheduling systems for automated spacecraft commanding. Projects include planning and scheduling for the first deep-space mission of NASA’s New Millennium Program, and for design trades analysis for the Pluto Express project. Gregg holds both a B.S. and M.S. degree in Computer Science from the University of Illinois where he specialized in Artificial Intelligence.*

***Steve Chien** is Technical Group Supervisor of the Artificial Intelligence Group of the Jet Propulsion Laboratory, California Institute of Technology where he leads efforts in research and development of automated planning and scheduling systems. He is also an adjunct assistant professor in the Department of Computer Science at the University of Southern California. He holds B.S., M.S., and Ph.D. degrees in Computer Science from the University of Illinois. His research interests are in the areas of planning and scheduling, operations research, and machine learning.*

Tobias Mann *was born in Spokane, Washington and is currently an undergraduate at the University of Washington in both the Computer Science and Philosophy departments. He has a wife and a two-year old son. His interests include planning and scheduling, machine learning, bicycling, and really good coffee.*

William "Curt" Eggemeyer *graduated from Washington University in St. Louis with a BA majoring in geology. In 1978, he became a JPL employee and began working on the Voyager project as a spacecraft sequence engineer. He demonstrated the applicability of utilizing artificial intelligence (AI) techniques to the sequence process with the generation of Voyager Uranus encounter sequences with a program called DEVISER, developed by Steven Vere, in 1983-1984. He codeveloped a prototype, Plan-IT, further advancing sequencing software tool concepts. From 1991-1992, he reworked Plan-IT into a more capable an robust sequencing tool, called Plan-IT-2, that is presently being used by DATA-CHASER, Galileo, and Mars Pathfinder projects.*

Jason Willis *is a currently pursuing a Master's Degree in Aerospace Engineering specializing in spacecraft systems from the University of Colorado Boulder, where he also received his B.S. in Aerospace engineering. He has worked at the Colorado Space Grant College for the past three years first as Electrical Integration Team Lead on the ESCAPE II shuttle payload the was launched on STS-66. He is currently the hardware systems engineer for the DATA-CHASER project.*

Sam Siewert *is a graduate research assistant with Colorado Space Grant College. He is working on a Ph.D. in Computer Science at the University of Colorado Boulder where he received his M.S. in Computer Science in 1993. He received his B.S. in Aerospace Engineering from the University of Notre Dame in 1989, worked three years for McDonnell-Douglas Astronautics Corporation in Guidance, Navigation and Control developing simulation, space environment models, and guidance systems software for the Space Station and the Aeroassist Flight Experiment. During that time, he also worked for McDonnell-Douglas at Johnson Space Center in the Shuttle Mission Control Center, developing shuttle ascent and entry monitoring and cockpit avionics visualization software, before returning to graduate school.*

Peter Stone *is a Ph.D. candidate in Computer Science at Carnegie Mellon University (CMU). He completed his undergraduate education in Mathematics with a concentration in Computer Science at the University of Chicago in 1993. His interests are in the areas of multiagent systems, collaborative and adversarial machine learning, and planning, especially in multiagent, real-time environments.*